Microservices Validation: Methodology and Implementation

Dmitry Savchenko and Gleb Radchenko

South Ural State University, Chelyabinsk, Russia
gleb.radchenko@susu.ru

Abstract. Due to the wide spread of cloud computing, arises actual question about architecture, design and implementation of cloud applications. The microservice model describes the design and development of loosely coupled cloud applications when computing resources are provided on the basis of automated IaaS and PaaS cloud platforms. Such applications consist of hundreds and thousands of service instances, so automated validation and testing of cloud applications developed on the basis of microservice model is a pressing issue. There are constantly developing new methods of testing both individual microservices and cloud applications at a whole. This article presents our vision of a framework for the validation of the microservice cloud applications, providing an integrated approach for the implementation of various testing methods of such applications, from basic unit tests to continuous stability testing.

Keywords: microservices · ontologies · cloud computing · validation

1 Introduction

The microservice model describes a cloud application as a suite of small independent services, each running in its own container and communicating with other services using lightweight mechanisms. These services are built around separate business capabilities, independently deployable and may be written by different development teams using different programming languages and frameworks [1]. We can mention several platforms (with varying degrees of completeness) that support the microservice model: VAMP [2], Mjolnirr [3] and Netflix Cloud Platform [4]. In the paper [5], the following features of microservices were described:

- *Open Interface* – microservice should provide an open description of interface and communication messages format (either API or GUI).
- *Specialization* – each microservice provides a support for an independent part of application's business logic.
- *Containerization* – isolation from the execution environment and other microservices based on a container virtualization approach. Technologies like OpenVZ, Docker or Rocket [6] became a de-facto standard for implementation of such approach.

– *Autonomy* – microservices can be developed, tested, deployed, destroyed, moved or duplicated independently and automatically. Continuous integration is the only option to deal with such development and deployment complexity.

One of the most important stages of the process of continuous integration is a continuous validation of the software. The goal of this study is to develop a microservice validation methodology and design a software solution for automated microservice systems testing support. To achieve this goal it is necessary accomplish the following tasks:

– provide a review of existing distributed systems validation methods, including cloud, multi-agent and actor-based systems;
– develop a microservice validation methodology;
– design a framework for microservice systems validation support.

In this article, we will describe our model of validation of microservice systems and it's implementation as a prototype of microservice systems validation framework.

The rest of this paper is organized as follows. In Section 2 we will provide a review of the current state of distributed systems testing and validation methods. In Section 3 we would describe the model of validation of microservice systems. In Section 4 we would propose an architecture and describe prospective approaches to implementation of microservice systems validation framework. In Section 5 we would summarize the results of our research and discuss further research directions.

2 Distributed Systems Validation

To develop a model of microservice systems validation, we should analyze existing testing methods that used for such distributed systems. In the paper [5] we proposed an approach to a microservice system validation based on ISO/IEC 29119 standard suite. Authors of [8] describe an approach to microservice testing based on BDD approach.

We also analyze the validation methods that are used in a set of related distributed systems models, like multi-agent systems and actors. For example, it is possible to apply ontologies-based multi-agent systems validation method [7] to provide a validation of microservices communication. However, there are some approaches that can't be easily implemented for the microservice model. For example, paper [9] describes the testing framework, which operates in accordance with the actors model. Such analysis can be carried out only for the actor systems, because, unlike the actor model, there is no standard requirement for any microservice that it should be able to create another instance of microservice.

We should also mention integration and stability testing methods are developed for highly loaded cloud applications. The "Chaos Monkey" [10] failure-injection framework, provided by Netflix, implements a permanent background

work process, which at random times introduces occasional failures in the production platform. This process may trigger a violation of network connectivity, crash random processes or entire units of the computing system. This testing approach leads to the fact that the application is designed with consideration of possible random failures, and if they occur, they do not cause application malfunction. Another approach, proposed by the Twitter development team in [11], provides automated web services validation based on mirroring of user requests incoming to the production environment and their simultaneous execution on the testing environment. Comparison of results of execution of these requests allows to evaluate the quality of the system being developed in the real flow of user requests, thus avoiding the inconsistency of a standard user activity processing.

The analysis of existing methods of testing of microservice systems shows, that there is no unified model of microservice system validation that would support a continuous integration of microservices.

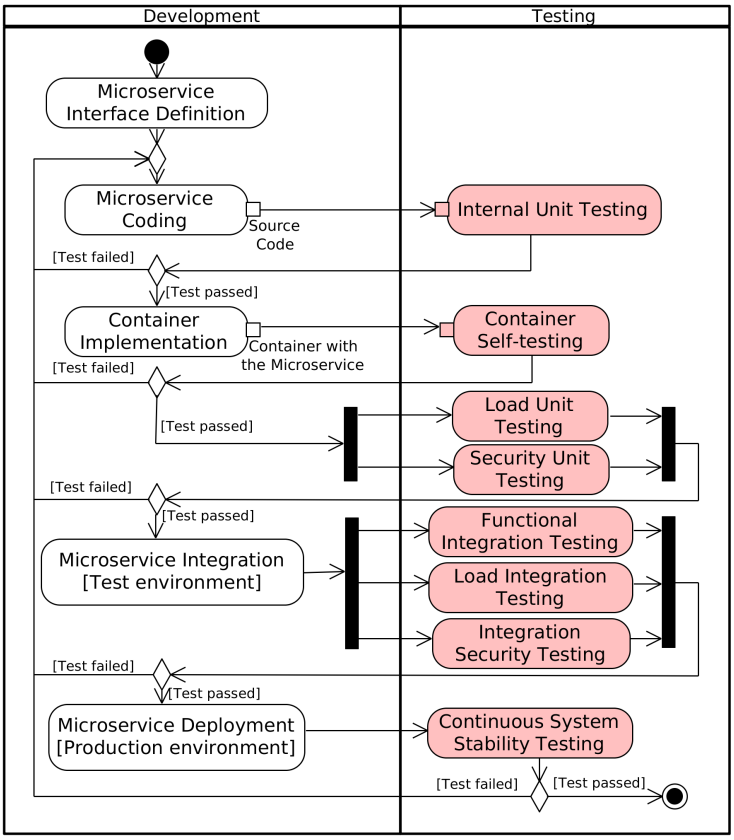

Fig. 1. Microservice validation procedure

3 Model of Validation of Microservice Systems

Based on the microservice features shown in the introduction, we propose the following model of microservice systems validation (see fig. 1).

1. To provide an automated validation of microservices interface and communication process, we should require a *definition of the interface* of every microservice.
2. According with the features of the programming language and framework chosen to implement the microservice, a developer should provide appropriate automated *unit-testing of the microservice source code* to ensure it's compliance with the requirements.
3. If the source code is passing all the unit tests, the microservice is packed into the container and a set of *container self-tests* is provided to ensure that all the components of the container are functioning correctly and the interface of the container corresponds to the interface definition provided on the step 1.
4. If the self-test is successful, then microservice *load testing and security testing* is provided. Steps 1-4 can be provided locally on a microservice developer's machine.
5. If all tests pass, the microservice is deployed on a test microservice environment, where *functional integration testing, load integration testing and security integration testing* is performed. It allow to detect a set of issues caused by microservices orchestration, including load balancing, life cycle management and communication issues.
6. If the microservice is passing all tests inside the test environment, it can be deployed to the production environment. A *continuous stability testing* is performed in a production environment in accordance with the methodology of deliberate provocation of random failures of the system components.

4 Microservice Systems Validation Framework

To evaluate the proposed validation model, a Microservice Systems Validation Framework (MSVF) is being developed. The MSVF should support microservice application testing from the source code to the continuous stability testing, regardless of the basic programming languages and software frameworks used for the microservice development (see fig. 2).

MSVF would provide its users a catalog of testing methods, that can be used to test a microservice application and an API for integration with continuous integration platforms. Each testing method can be considered as a separate microservice template, that can be tuned to provide validation of specific part or activity of user's microservice application.

We can define the following main actors, who would interact with the MSVF:

– *Validation methods developer*: provides development of new validation methods and their integration into the MSVF. The validation method can be implemented as an independent microservice, that implements a specific

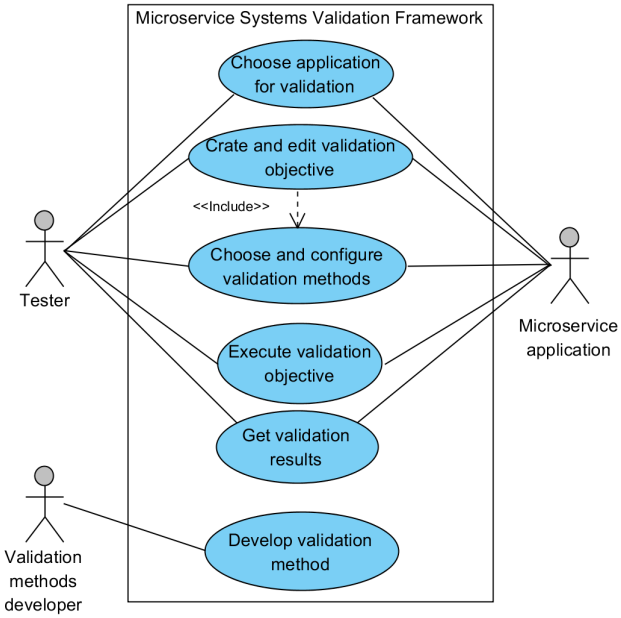

Fig. 2. Microservice Systems Validation Framework Use Cases

method of validation that can be adopted to a specific microservice or microservice application, including:

- *component-level testing*: supports unit-testing of the microservice source code and container self-testing;
- *integration-level testing*: supports functional integration testing of the microservice system on a testing environment;
- *load testing* : supports load integration testing of the microservice system on a testing environment;
- *stability testing* : supports validation methods that provides deliberate provocation of random failures of the microservice system in a production environment.

– *Tester*: is the main user of the MSVF platform. Tester can choose a service or microservice application for validation; define a validation objective that consists of a set of validation methods, adapted to the chosen application; execute validation objectives and gather the results of the validation.
– *Microservice application*: represents the validated microservice application.

We can define the following basic components of the MSVF (see fig. 3):

– *Validation Methods Manager* responds for listing, creation and modification of validation methods available in the MSVF.

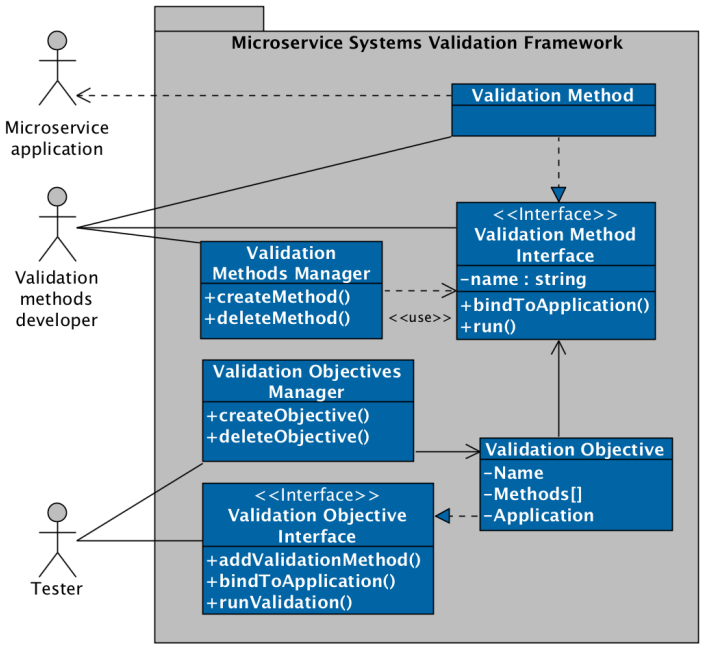

Fig. 3. Microservice Systems Validation Framework Architecture

- *Validation Method Interface* is a common API that should be provided by any validation method in the MSVF to support adaptation of validation method to a specific microservice application.
- *Validation Method* is an implementation of specific validation method in the form of microservice template.
- *Validation Objectives Manager* responds for listing, creation and modification of validation objectives.
- *Validation Objectives Interface* defines an interface of validation objectives, including methods for integration and tuning of validation methods.
- *Validation Objective* represents a suite of validation methods, tuned for a specific microservice application.

Currently, the MSVF is being developed to support VAMP and Mjolnirr microservice platforms and Jenkins as a continuous integration framework. VAMP and Mjolnirr cloud infrastructure is deployed on a set of computing nodes provided by the Supercomputer Center of South Ural State University. The MSVF is being implemented as a separate Java-based microservice with its own web and REST interface, providing integration and execution of validation methods and objectives.

To support the validation process, a set of standard validation methods is being developed and integrated to the MSVF, including component-level, integration-level, load and stability testing methods.

5 Conclusion

In this paper, we provided an overview of existing validation approaches that used to test distributed and cloud systems. On the basis of this overview, we presented a model of validation of microservice systems, supporting such microservice features as open interface, containerization and autonomy. This methodology covers the microservice systems development process, from the creation of a separate microservice to the production environment continuous stability testing. Based on this model, an architecture of microservice systems validation framework was presented.

Currently we implement the microservice systems validation framework as a software solution that would support microservice applications validation in the cloud environment. It would provide its users a catalog of microservice applications testing methods and an API for integration with continuous integration platforms. Currently, the microservice systems validation framework is being developed to support VAMP and Mjolnirr microservice platforms and Jenkins as a continuous integration framework.

Acknowledgment

The reported study was partially supported by RFBR, research project No. 14-07-00420-a and by Grant of the President of the Russian Federation No. MK-7524.2015.9.

References

1. Thones, J.: Microservices. IEEE Softw. 32, 116–116 (2015).
2. Vamp : The Very Awesome Microservices Platform, http://vamp.io/.
3. Savchenko, D., Radchenko, G.: Mjolnirr: A Hybrid Approach to Distributed Computing Architecture and Implementation. 4th International Conference on Cloud Computing and Services Science. 445–450, Barcelona, Spain (2014).
4. Tilkov, S.: The Modern Cloud-Based Platform. IEEE Softw. 32, 116–116 (2015).
5. Savchenko, D., Radchenko, G.: Microservices validation: Mjolnirr platform case study. 38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO'2015. 248–253. IEEE, Chroatia, Opatija (2015).
6. Pahl, C.: Containerization and the PaaS Cloud. IEEE Cloud Comput. 2, 24–31 (2015).
7. Nguyen, C., Perini, A., Tonella, P.: Ontology-based Test Generation for MultiAgent Systems. 7th international joint conference on Autonomous agents and multiagent systems. 1315–1320. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2008).

8. Rahman, M., Gao, J.: A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development. 2015 IEEE Symposium on Service-Oriented System Engineering (SOSE). 321–325. IEEE (2015).
9. Tasharofi, S., Karmani, R., Lauterburg, S., Legay, A., Marinov, D., Agha, G.: TransDPOR: A Novel Dynamic Partial-Order Reduction Technique for Testing Actor Programs. Formal Techniques for Distributed Systems. 219–234 (2012).
10. Izrailevsky, Y., Tseitlin, A.: The Netflix Simian Army, http://techblog.netflix.com/2011/07/netflix-simian-army.html.
11. Khanduri, P.: Diffy: Testing services without writing tests, https://blog.twitter.com/2015/diffy-testing-services-without-writing-tests.